What is claimed is:

1. A method for scheduling tasks for processing in a coprocessor, comprising:

generating a run list comprising a list of tasks for processing by a coprocessor, wherein the run list is generated by a central processing unit (CPU);

delivering the run list to a scheduler process, wherein the scheduler process prepares the tasks on the run list for processing by a coprocessor; and

processing the tasks by the coprocessor in an order indicated by the run list, wherein if a switching event occurs when processing one of the tasks, the coprocessor switches immediately to the next task on the run list.

2. A method according to claim 1 wherein the coprocessor is a graphics processing unit (GPU).

3. The method of claim 1 wherein a switching event comprises at least one of a completion of processing a previously submitted task, a page fault in processing a task, a general protection fault in processing a task, and a request by the CPU to switch to a new run list.

4. The method of claim 1, further comprising signaling the CPU with an interrupt signal when the coprocessor switches from one task in the run list to the next task in the run list, wherein when the CPU receives the interrupt signal, the CPU builds a new run list for the coprocessor.

5. The method of claim 1, further comprising generating a second run list by the scheduler process, whereby the scheduler process can initiate a change in the order of tasks to be processed by the coprocessor.

6. The method of claim 5, further comprising prohibiting a first task of a first run list from appearing in the second run list.

7. The method of claim 5, further comprising prohibiting a second task of a first run list from appearing in the second run list unless it is a first task of the second run list.

8. The method of claim 1, further comprising storing information regarding the history of coprocessor switches from task to task in a specified system memory location readable by the scheduler.

9. The method of claim 8 wherein the system memory location readable by the scheduler is a history buffer available to a single coprocessor only.

10. The method of claim 8 wherein the history buffer comprises sufficient memory to store at least twice the amount of information required to store the run list.

11. The method of claim 8, further comprising specifying a coprocessor write pointer, which indicates a location in the history buffer where the coprocessor can write new information.

12. A computer readable medium comprising computer executable instructions for carrying out the method of claim 1.

13. A modulated data signal carrying computer executable instructions for use in performing the method of claim 1.

14. A computing device comprising means for performing the method of claim 1.

15. A method for streamlining operations in a coprocessor by indicating whether all required memory resources for a task are available prior to beginning the processing of the task, comprising:

preparing a task for processing in a coprocessor by paging memory resources associated with the task into coprocessor-readable memory;

sampling the memory resources to determine if all required memory resources are in a proper location in the coprocessor-readable memory;

recording whether all required memory resources are in a proper location in the coprocessor-readable memory, wherein said recording generates an indicator memory resource that is associated with the task;

processing said indicator memory resource substantially at the beginning of processing the task, wherein if said indicator resource indicates that all required memory resources are not in a proper location in the coprocessor-readable memory, the coprocessor stops processing the task.

16. A method according to claim 15 wherein the coprocessor is a GPU.

17. A method according to claim 15 wherein the task is represented by a DMA buffer.

18. A method according to claim 15 wherein the coprocessor stops processing the task because processing said indicator memory resource generated a page fault.

19. A method according to claim 15, further comprising maintaining a list of tasks that the coprocessor stopped processing, so that all required memory resources can be brought to a proper location in coprocessor-readable memory at a later time.

20. A method according to claim 19 wherein the later time is determined based on a priority of tasks on the list of tasks.

21. A method according to claim 20, further comprising a periodic priority boost that increases the priority of one or more tasks on the list of tasks to ensure that all tasks eventually can be processed.

22. A method according to claim 15, further comprising generating a page fault when a context switch occurs to a context that references an invalid ring buffer or an invalid DMA buffer.

23. A set of instructions for controlling the flow of instructions in a DMA stream that is processed on a coprocessor, comprising:

a fence instruction, including both a piece of data and an address, that can be inserted into a DMA stream such that when the fence instruction is read by a coprocessor from the DMA stream, the coprocessor writes the piece of data associated with the fence instruction at the address;

a trap instruction that can be inserted in a DMA buffer, such that when the coprocessor reads the trap instruction, the coprocessor generates a CPU interrupt; and

an enable/disable context switching instruction, such that the coprocessor will permit switching away from a current coprocessor context when the enable/disable context switching instruction instructs the coprocessor to enable context switching, or not permit switching away from a current coprocessor context when the enable/disable context switching instruction instructs the coprocessor to disable context switching.

24. A Graphics Processing Unit (GPU) that is configured to support the instructions of claim 23.

25. The instructions of claim 23 wherein the fence instruction, when read by a coprocessor from the DMA stream, first causes the coprocessor to ensure that any pixels from primitives preceding the fence instruction have been retired and properly written to memory.

26. A Graphics Processing Unit (GPU) that is configured to support the instructions of claim 25 wherein the GPU continues processing primitives following the fence instruction while the GPU is waiting on the last pixel of the instruction before the trap instruction to be retired.

27. The instructions of claim 23 wherein the trap instruction, when read by a coprocessor from the DMA stream, first causes the coprocessor to ensure that any pixels from primitives preceding the trap instruction have been retired and properly written to memory.

28. A Graphics Processing Unit (GPU) that is configured to support the instructions of claim 25 wherein the GPU continues processing primitives following the trap instruction while the GPU is waiting on the last pixel of the instruction before the fence instruction to be retired.

29. The instructions of claim 23 wherein when the enable/disable context switching instruction instructs the coprocessor to enable context switching, additional instructions are employed to ensure one of that only privileged DMA buffers will be processed by the coprocessor, that no context switching instructions will be present in a coprocessor DMA stream, that a coprocessor DMA stream will not run out of instructions, and that no page faults will occur.

30. A scheduler mechanism for use in conjunction with a coprocessor, wherein the scheduler mechanism implements high level synchronization objects by carrying out a method comprising:

generating a wait instruction comprising a fence, wherein a fence is an instruction containing both a piece of data and an address that is inserted in a DMA stream, such that when the address is read by a coprocessor, it will cause the coprocessor to write the piece of data associated with the fence at a specified location; and

protecting a section of a DMA buffer from executing until the section of a DMA buffer is rescheduled by a CPU, wherein the CPU will reschedule a section of a DMA buffer after the wait instruction is processed.

31. A scheduler mechanism for carrying out the method of claim 30, said method further comprising refraining from submitting one or more DMA buffers that would logically follow the fence into a ring of a coprocessor context until a wait condition is satisfied.

32. A scheduler mechanism for carrying out the method of claim 30, said method further comprising moving a coprocessor context to a wait list for an object until the object is signaled, wherein signaling an object comprises a fence followed by an interrupt command in a coprocessor context DMA stream.

33. A coprocessor for supporting the seamless display of graphics for applications by carrying out the steps of a method comprising:

allocating data corresponding to a first display surface contiguously in a coprocessor-readable memory location, wherein a first base address is used by a display device to identify a location of said data corresponding to a first display surface;

allocating data corresponding to a second display surface contiguously in a coprocessor-readable memory location; and

generating a flip instruction that reassigns said first base address so that the display device instead identifies the location of a second base address, wherein said second base address refers to said data corresponding to a second display surface, and wherein said flip instruction is generated by a coprocessor.

34. A coprocessor for carrying out the method of claim 33, said method further comprising processing the flip instruction, wherein the flip instruction reassigns said first base address immediately.

35. A coprocessor for carrying out the method of claim 33, said method further comprising processing the flip instruction, wherein the flip instruction reassigns said first base address upon the occurrence of a subsequent display synchronization period.

36. A computer readable medium containing instructions for carrying out the steps of claim 33.

37. A coprocessor for carrying out the method of claim 33, wherein said coprocessor is a GPU.